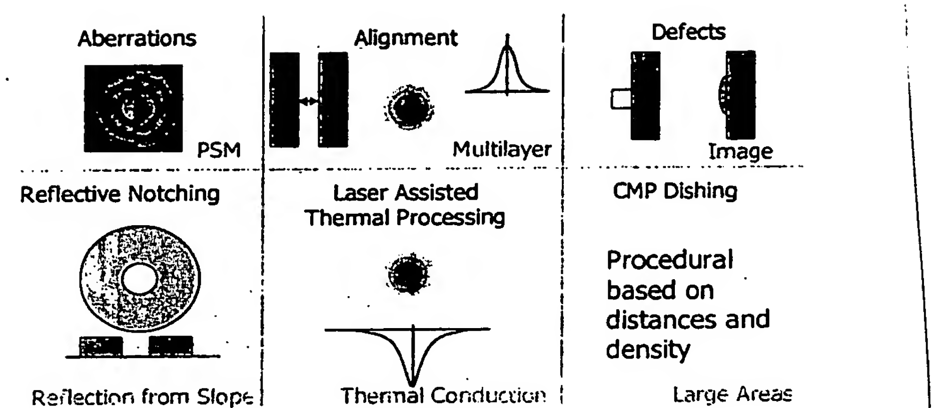


Figure 1: Pattern matcher test patterns for various applications.

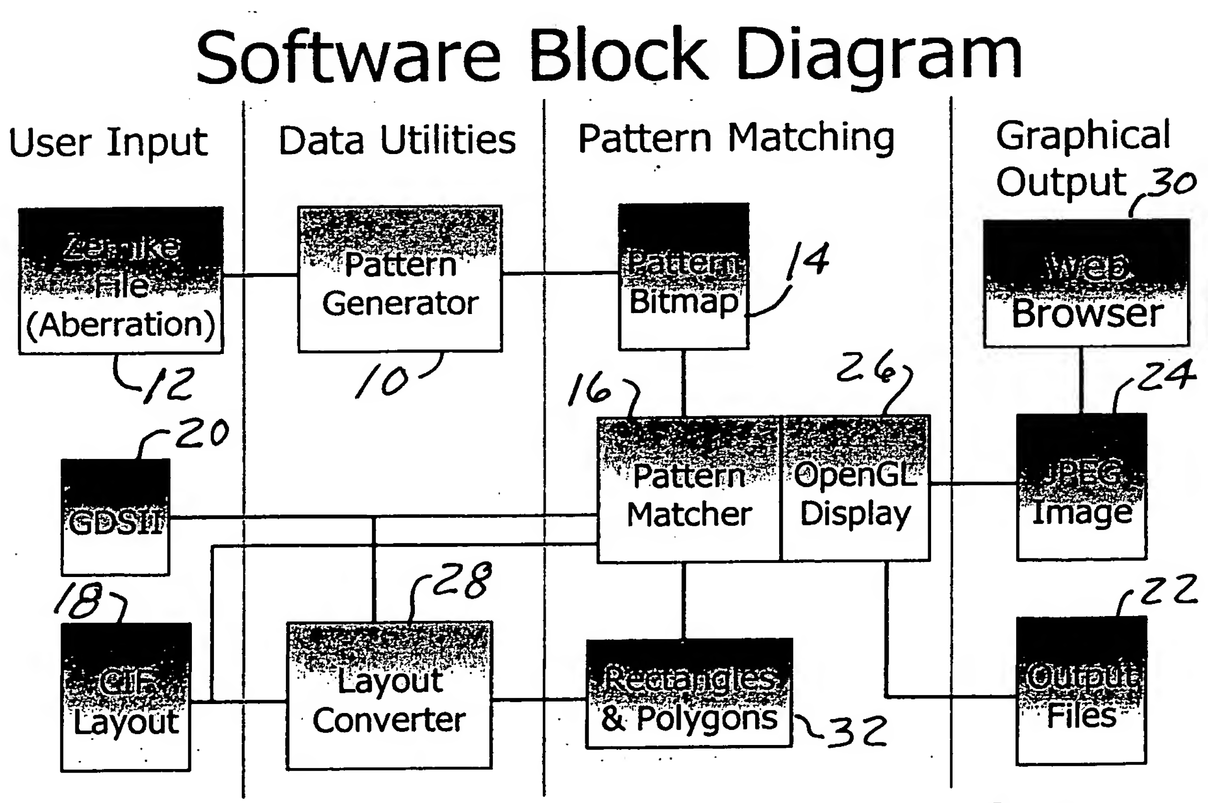


FIG. 2A

BEST AVAILABLE COPY

READ LAYOUT AND PATTERN (40)

PRE-INTEGRATE PATTERN (42)

PARTITION LAYOUT (44)

SPLIT POLYGONS IN PARTITION INTO RECTANGLES/TRIANGLES (46)

SORT RECTANGLES/TRIANGLES IN PARTITION INTO REGIONS BY POSITION (48)

RECTANGE OVERLAP REMOVAL, MERGE, LAYER BOOLEANS IN PARTITION, REGION (50)

EXTRACT EDGES AND CORNERS FROM REGTANGLES/TRIANGLES (52)

COMPUTE MATCH FACTORS (54)

SORT RESULTS IN ALL PARTITIONS (56)

DISPLAY/PRINT RESULTS (58)

MODIFY LAYOUT, REPEAT COMPUTATION OF MATCH FACTORS (60)

FIG. 2B

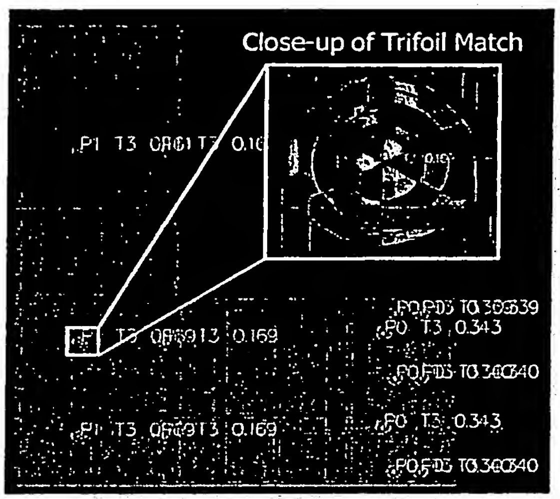


Figure 3: Trifoil and coma patterns matched on 0/180-degree FPGA interconnect layout.

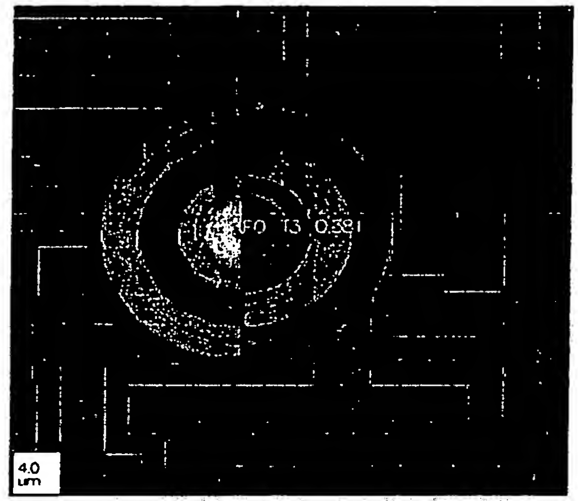


Figure 4: Coma pattern match on two-layer mask layout with 45-degree edges. The white square is 4µm.

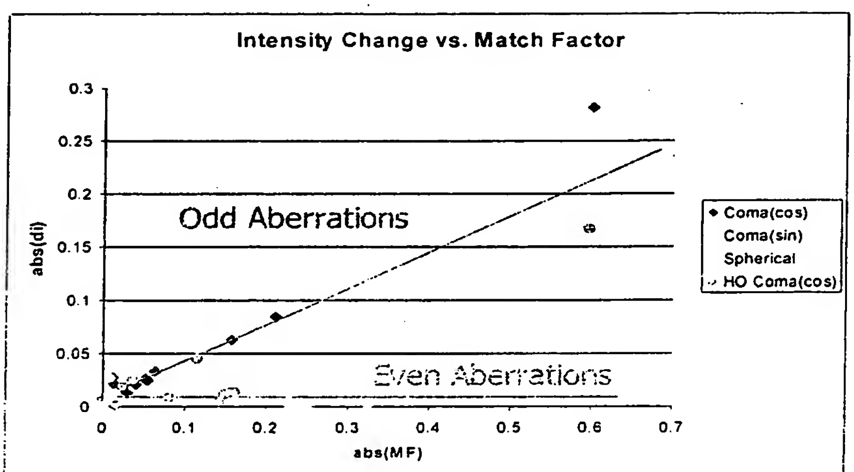


Figure 5: Simulated intensity change vs. match factor prediction for various aberration patterns and layouts.

Generic Pattern Matching Code

1. Divide input shapes (polygons) into geometric primitives

2. Spatially organize primitives by x, y, etc.

3. Compute Match Factor (MF):

```
for each pattern P
    for each orientation of P
        for each match type T
            for each X,Y match location
                for each geom. Primitive G overlapping P
                    add contribution of G on P at X,Y to MF
```

Time dominated by #3: #patterns x #orientations x #types x #locations x #primitives_overlap_pattern x time(primitive)

FIG. 6

Data Structures

- Input = polygons, rectangles (special case of a polygon), paths (can be converted to polygons), and circles (can be approximated by many-sided polygons) = polygons
- Geometric Primitives:

Type	Number in layout	Operations to add to MF (time)
Pixel (Bitmap Alg.)	Very Large (area)	1
Edge Intersection	Large (perimeter)	2
Rectangle	Medium	4
Triangle	Small (or none)	4 to 12 (if split)

- Higher-level primitives (lower in table) are much more efficient to store and use

FIG. 7

Polygon Splitting (Bitmap)

- Manhattan Polygon => Bitmap
  - Too many pixels to store – large blocks of the same value

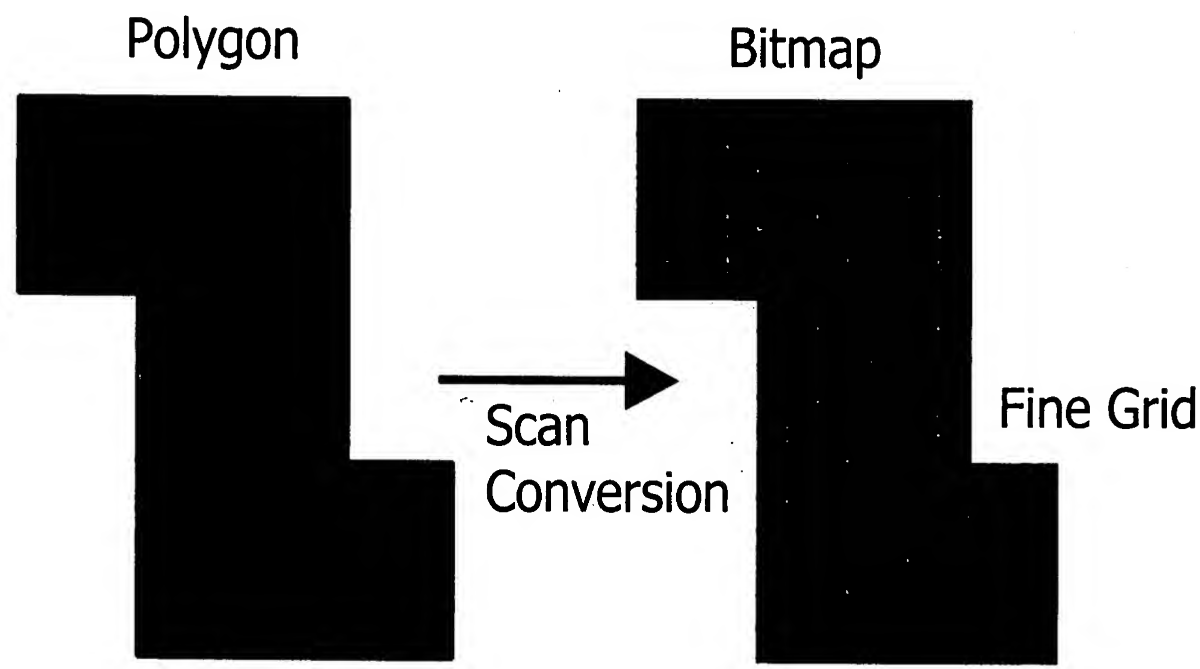


FIG.8

Polygon Splitting (Edges)

- Manhattan Polygon => Edges
  - Well, actually rectangle strips between 2 edges

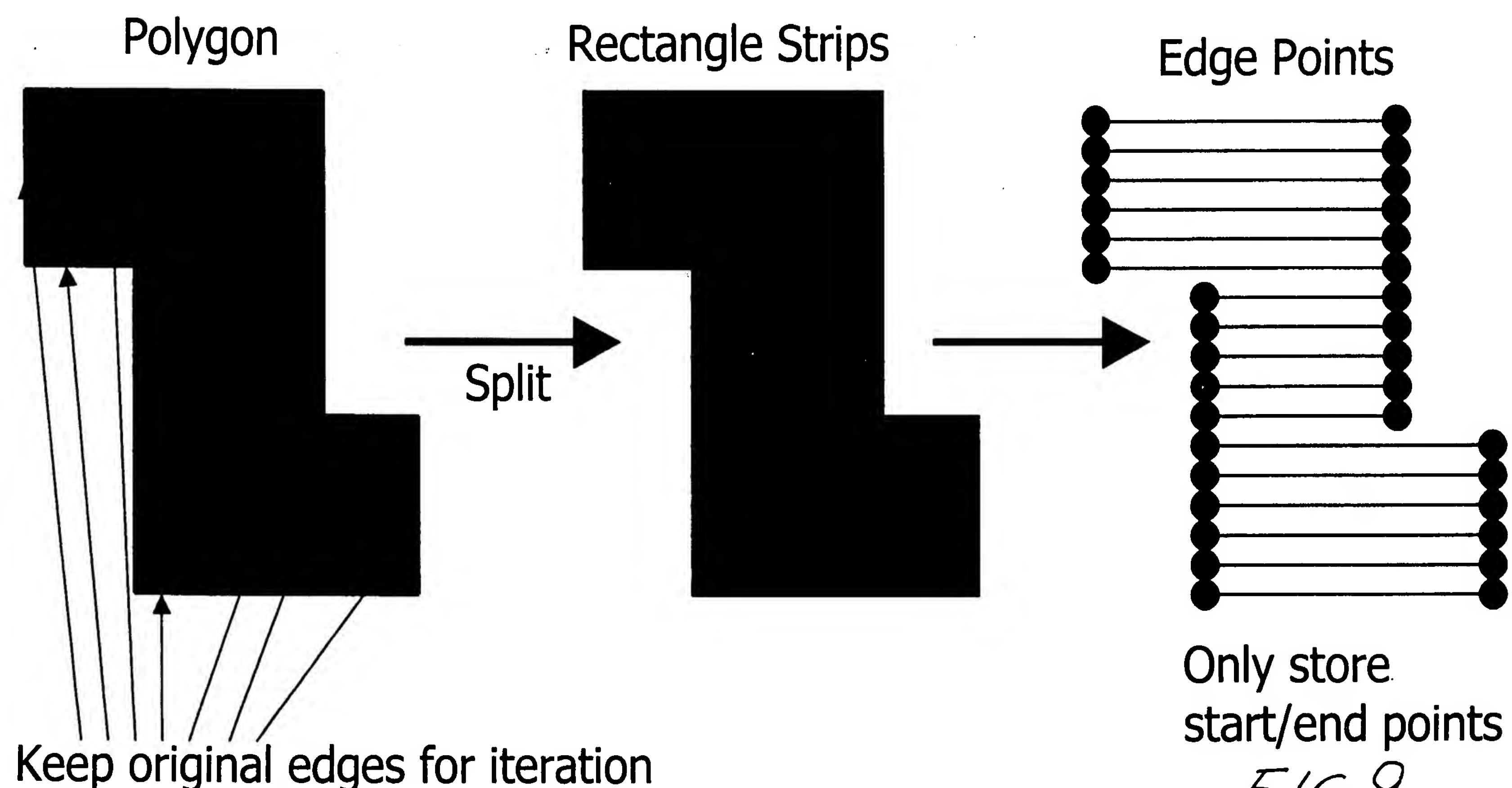
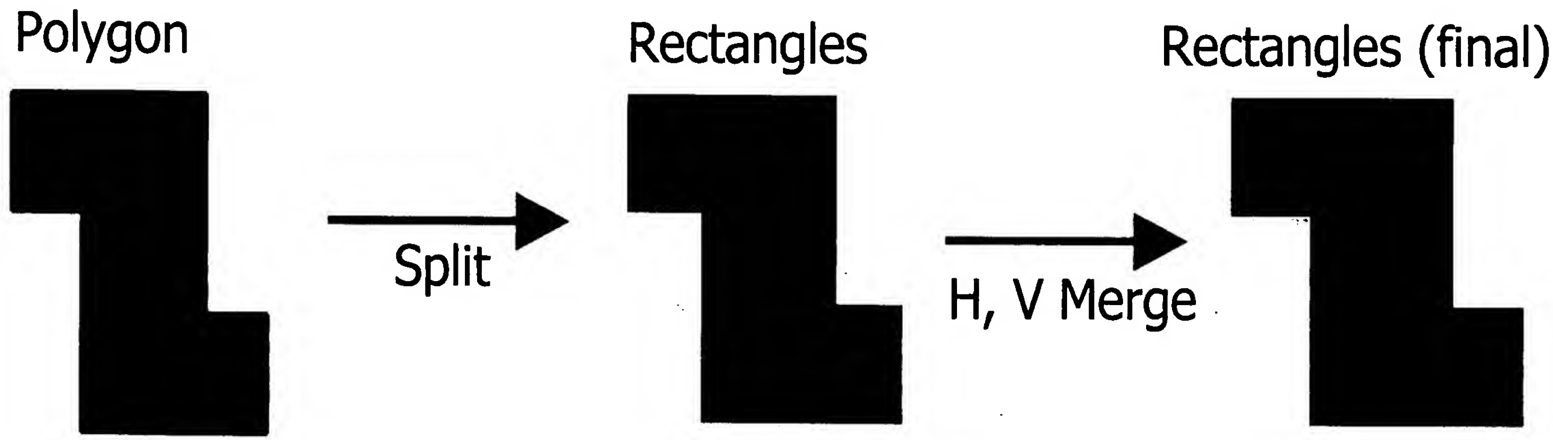


FIG. 9

Polygon Splitting (Rectangles)

- Manhattan Polygon => Rectangles A



- Non-Manhattan Polygon => Rectangles B

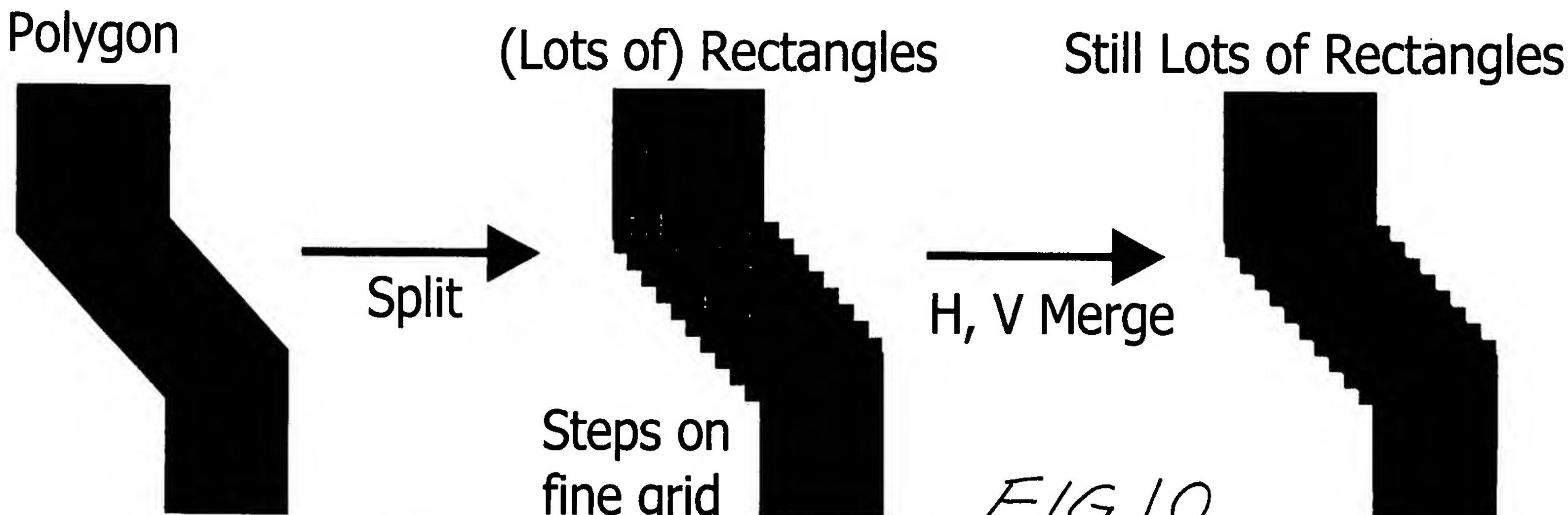


FIG. 10

Polygon Splitting (Triangles)

- Non-Manhattan Polygon => Rectangles + Right Triangles

Primary Goal: Min # Triangles
Secondary Goal: Min # Rectangles

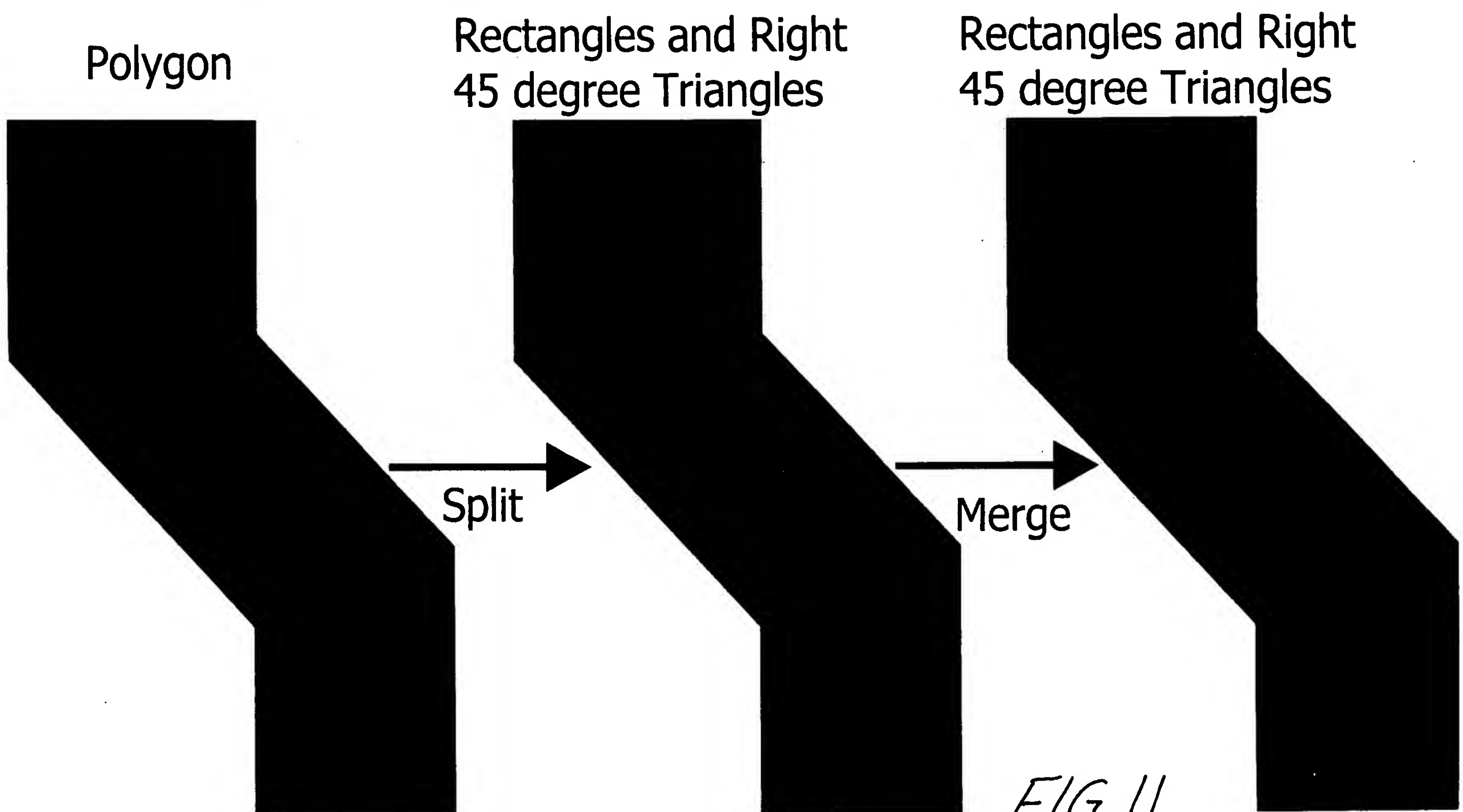


FIG. 11

Pattern Pre-Integration

- 1D Pre-Integration
  - Can be horizontal or vertical, either will work
  - Pre-integrated value = sum of all pattern values at and to the right

Pattern values	0	1	2	1	3	0	1
Pre-int values	8	8	7	5	4	1	1

- 2D Pre-Integration
  - Starts with 1D pre-integration
  - Pre-integrated value = sum of all pattern values at and to the right AND above (top right = orientation P0)

Typical PM pattern is 128x128

Pattern Values

PV

0	1	2	1
3	0	1	2
4	1	1	2
2	2	0	0

right →

1D Pre-Int to the right

IR (IU)

4	4	3	1
6	3	3	2
8	4	3	2
4	2	[illegible]	0

above →

2D Pre-Int top right

P0

4	4	3	1
10	7	6	3
18	11	9	5
22	13	9	5

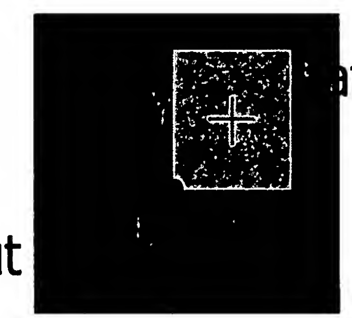
FIG. 12

Algorithm 1: Bitmap

- Entire layout represented as one huge bitmap of layers (like images on a computer screen)
- One rectangle is added at a time to the bitmap
- At every match location (edge, corner, etc.), each pattern pixel is multiplied by the layout pixel and summed:

$$MF(i+\frac{X}{2}, j+\frac{Y}{2}) = norm * \sum_Y \sum_X Layout(x+i, y+j) * Pat(x,y)$$

- Pattern size (X by Y) is typically 128x128 = 16384 ops



Layout

FIG. 13

Algorithm 2: Edge Intersections

- Store only the pixels along edges
- Run-length encoding in 1D – skip large runs of the same pixel value (rectangle strips)
- Pre-integrate pattern in 1D: $val(i,j) = \sum_{k=i}^{X} pat(k,j)$ for x intersection case
- Add MF contributions from each rectangle strip between two edges (either X or Y dir)

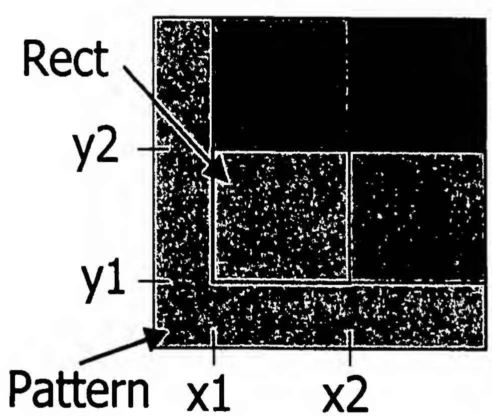
pat(...,j)	0	1	2	1	3	0	1
val(...,j)	8	8	7	5	4	1	1
r strip (weight 1)		1				-1	
		+	edges			-	

Contribution:
1*8 + (-1)*1 = 7

FIG. 14

Algorithm 3: Rectangles

- Simplest data structure: Store only the rectangles and pointers to them
- 2D encoding – only rectangle corners are needed
- Pattern integrated in 2D, rectangle LL corner clipped to pattern area
- Integrated pattern value is sum of values above and to the right: $val(i,j) = \sum_{k=i}^{Y} \sum_{l=j}^{X} pat(k,l)$



Contribution from rect at (x1,y1), (x2,y2) =
val(x1,y1) – val(x2,y1) – val(x1,y2) + val(x2,y2)

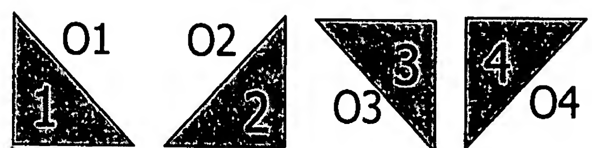
FIG. 15

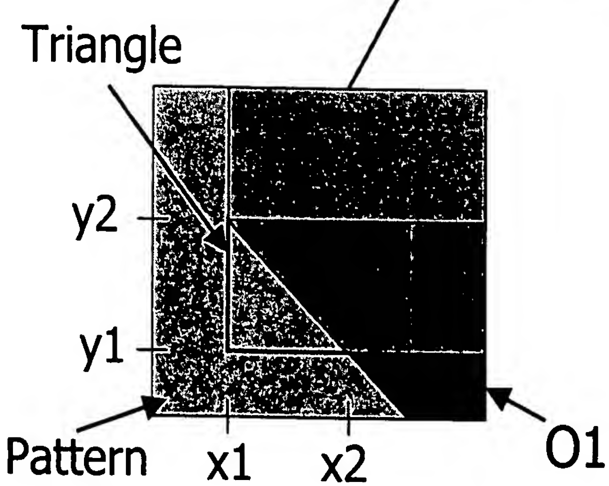
Only process LL corner and other 3 if inside pattern

Algorithm 3b: Triangles

- Extension of rectangle algorithm
- Pre-integration time/storage proportional to the number of unique angles
  - Limited to multiples of 45-degree angles in practice
    - 0, 45, 90, 135, 180, 225, 270, 315 deg => 8 pre-integrations

4 Orientations:





Need to pre-integrate in 8 directions for n*45 degree angled right triangles

Triangle clipping is difficult

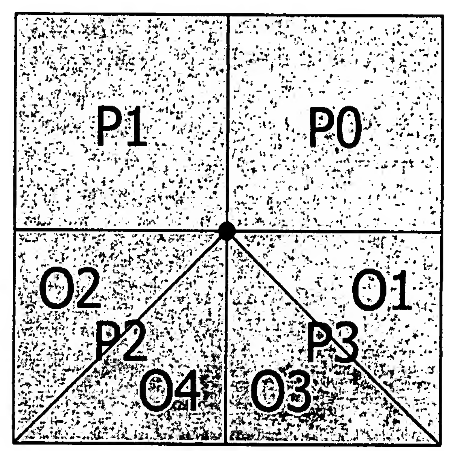


FIG. 16

Rectangle/Triangle Clipping

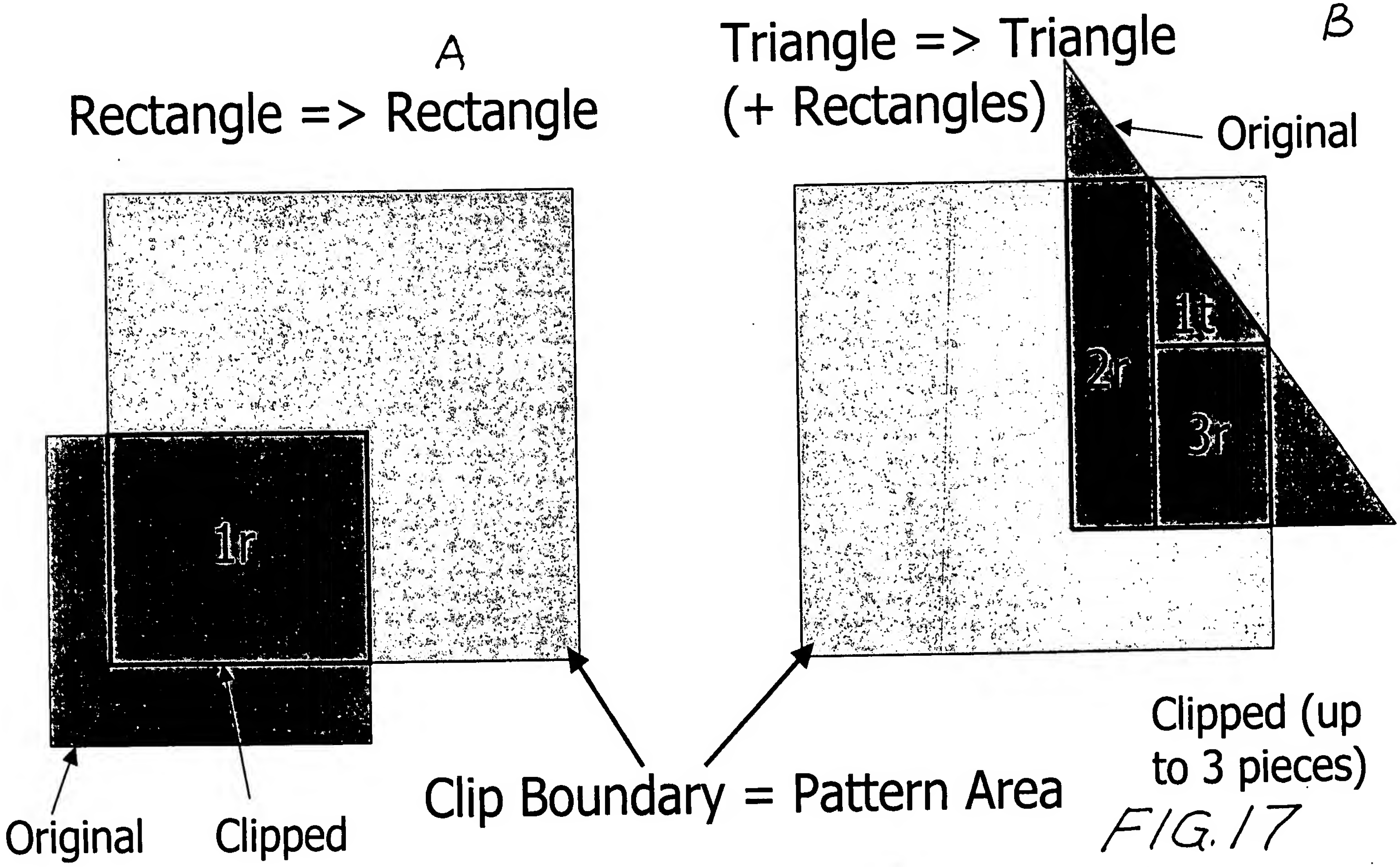
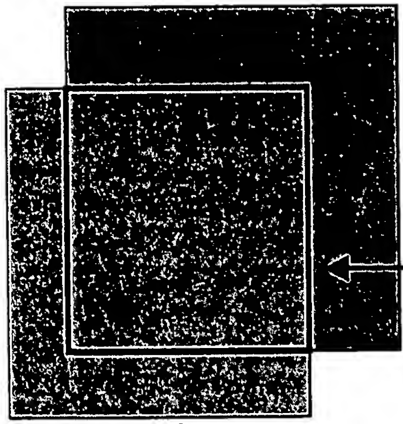


FIG. 17

Examples

Pattern



Input Rectangle

Area contributing to Match Factor

RL = rectangle length (3)
RH = rectangle height (3)
TL = triangle length (3)
TH = triangle height (3)

Bitmap Algorithm A

Pattern Values

PV

0	1	2	1
3	0	1	2
4	1	1	2
2	2	0	0

Pre-Integrate

(3+0+1) + (4+1+1) + (2+2+0) = 14
RL*RH = **9** Operations

Edge Intersection B

1D Pre-Int to the right

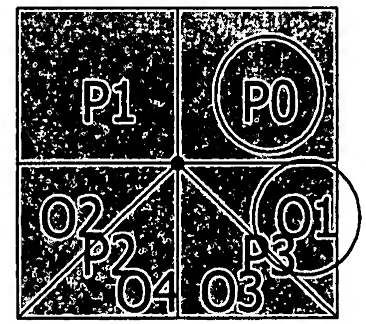
IR

4	4	3	1
6	3	3	2
8	4	3	2
4	2	0	0

(6-2) + (8-2) + (4-0) = 14
2*RH = **6** Operations

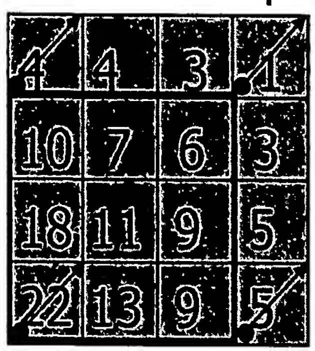
FIG. 18

Examples



Rectangle Algorithm

2D Pre-Int top right



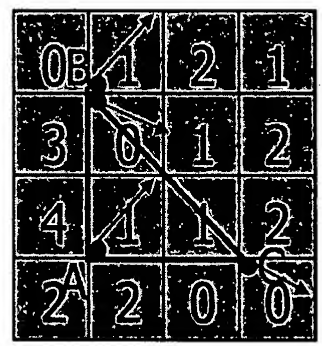
P0

LLC – ULC – LRC + URC =
22 – 4 – 5 + 1 = 14
Always **4** Operations

45-Triangle Algorithm

8-way Pre-Int → Precomputed:
P0 from rect algorithm

Pattern
Values



PV

O1(B) = 1+2+2+
(0+1+0)/2 = 5.5

O1(C) = 0/2 = 0

P0(A) – P0(B) – O1(B) + O1(C) =
11 – 4 – 5.5 + 0 = 1.5
4 Operations/Shape (12 max)

FIG. 19

Examples

1D Pre-Int to the right

IR

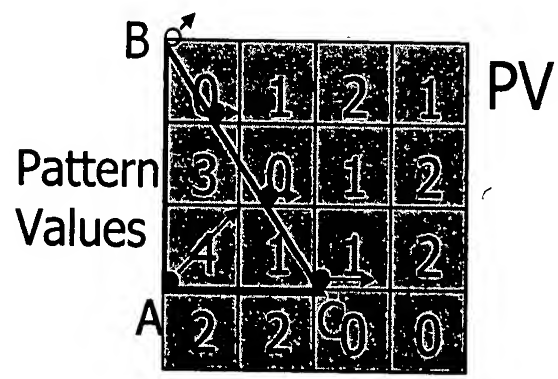
4	4	3	1
6	3	3	2
8	4	3	2
4	2	0	0

2D Pre-Int top right

P0

4	4	3	1
10	7	6	3
18	11	9	5
22	13	9	5

Non-45 degree Triangle (Proposed)



P0(A) – P0(B) – IR(B...C) =
18 – 0 – (4 + 3 + 3) = 8
TH + 2 = **5** Operations

Similar to edge intersection algorithm but reduced storage

FIG. 20

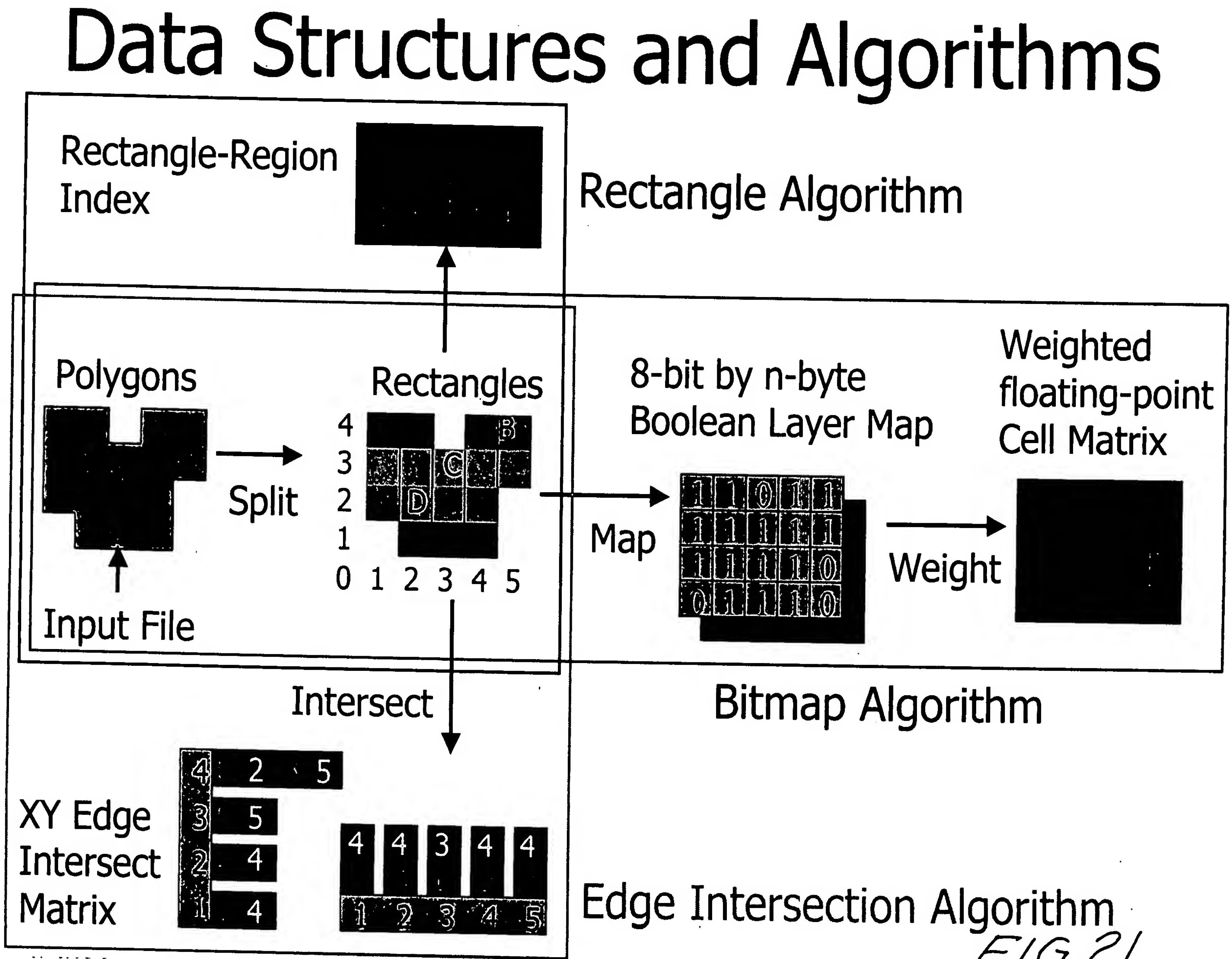

Data Structures and Algorithms
Rectangle-Region Index
Rectangle Algorithm
Polygons
Split
Input File
Rectangles
4
3
2
1
0 1 2 3 4 5
B
C
D
Map
8-bit by n-byte Boolean Layer Map
1 1 0 1 1
1 1 1 1 1
1 1 1 1 0
0 1 1 1 0
Weight
Weighted floating-point Cell Matrix
Bitmap Algorithm
Intersect
XY Edge Intersect Matrix
4 2 5
3 5
2 4
1 4
4 4 3 4 4
1 2 3 4 5
Edge Intersection Algorithm

FIG. 21